Exam 1 - A

Name:

Penn State access ID (xyz1234) in the following box:

Student ID number (9XXXXXXXX):

TA and/or section time:

Instructions:

- Answer all questions. Read them carefully first. Be precise and concise. Handwriting needs to be neat. Box numerical final answers.
- Please clearly write your name and your PSU access ID (i.e., xyz1234) in the box on top of **every page**.
- Write in only the space provided. You may use the back of the pages only as scratch paper. **Do not write your solutions in the back of pages!**
- **Do not write outside of the black bounding box**: the scanner will not be able to read anything outside of this box.
- We are providing two extra page at the end if you need extra space. Make sure you mention in the space provided that your answer continues there.

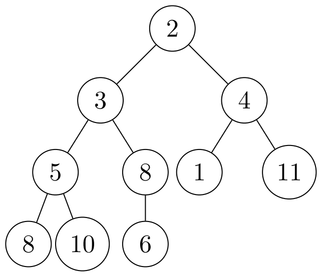
Good luck!

Name:	**PSU Access ID (xyz1234):**

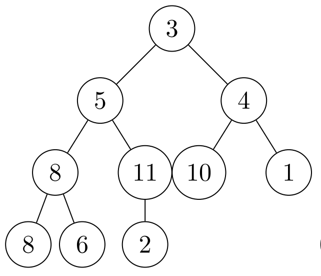
Multiple choice questions *(27 points)*

For each of the following questions, select the right answer by filling the corresponding grading bubble grading bubbles.

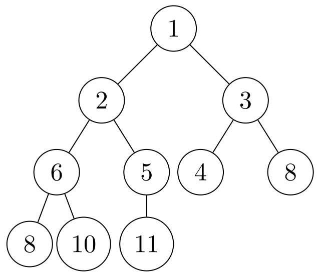
1. Run Build-Heap on the array $[3, 5, 4, 8, 6, 2, 10, 1, 8, 11]$ to construct a min heap. Select the resulting min heap.



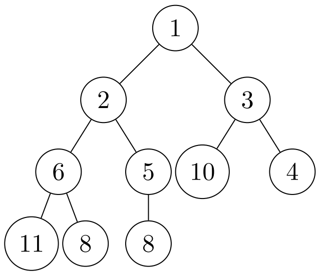
(a)



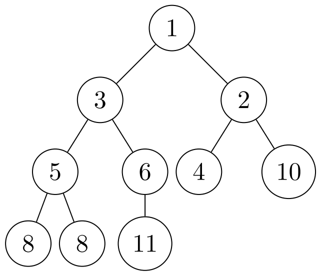
(b)



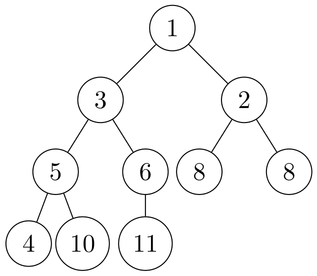
(c)



(d)



(e)



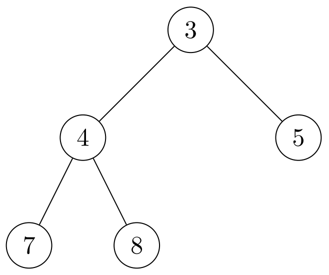
(f)

Answer.

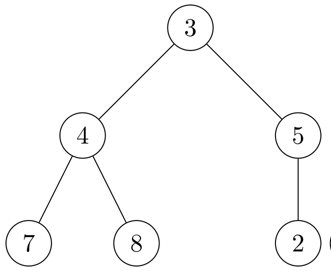
- ◯ (a)
- ◯ (b)
- ◯ (c)
- ◯ (d)
- [✓ (e)]
- ◯ (f)

Name:	**PSU Access ID:**

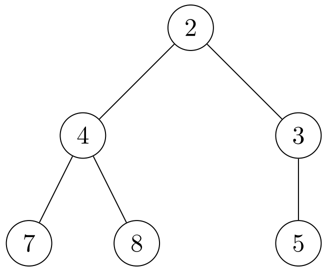
2. Consider the min heap below. Suppose the number 2 is inserted into the heap.



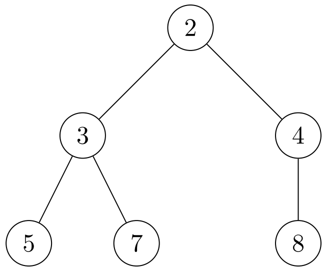
Which one of the following is the resulting min heap?



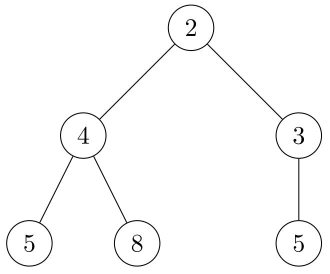
(a)



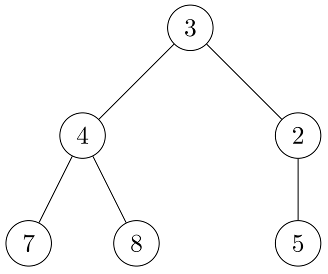
(b)



(c)



(d)



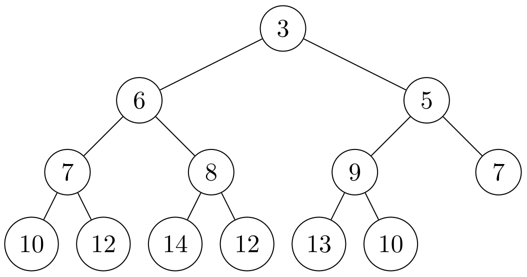
(e)

Answer.

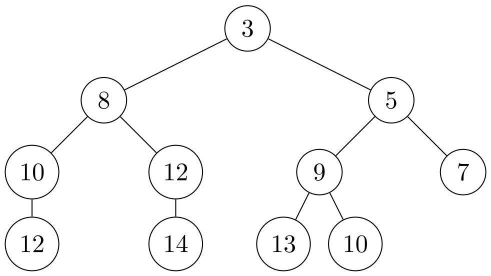
- ◯ (a)
- ● (b)
- ◯ (c)
- ◯ (d)

Name:	**PSU Access ID:**

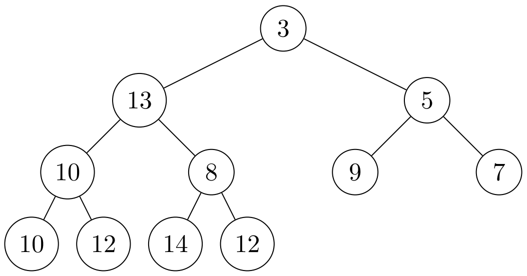
3. Consider the min heap on the left below. Suppose that first the element at position 3 of the array is removed from the heap (remember that we use 0 indexed arrays), and after that the element at position 1 is removed from the heap.



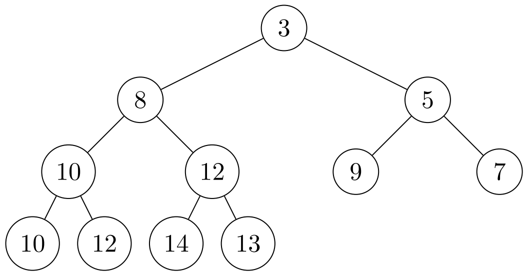
Which one of the following is the resulting min heap?



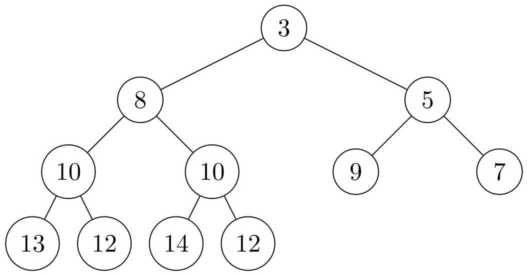
(a)



(b)



(c)



(d)

Answer.

- ◯ (a)
- ◯ (b)
- [✓ (c)]
- ◯ (d)
- ◯ (e)

Name:	**PSU Access ID:**

4. The array [2, 4, 3, 8, 9, 12, 13] corresponds to a min heap. In the space provided below, write down the resulting heap (as an array) after the first two iterations of the heapsort algorithm.

   Answer.

   ○ [2, 4, 3, 8, 9, 12, 13]
   ○ [12, 4, 13, 8, 9, 3, 2]
   ○ [2, 3, 4, 8, 9, 12, 13]
   ○ [8, 9, 12, 13, 4, 3, 2]
   [✓ [4, 8, 12, 13, 9, 3, 2]]
   ○ [13, 4, 12, 8, 9, 3, 2]

5. How many integer multiplications does Strassen's algorithm need to do to multiply two 4×4 matrices of integers?

   Answer.

   ○ 9
   ○ 8
   ○ 64
   [✓ 49]
   ○ 16
   ○ 7

6. Suppose that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$. Which of the following must be true?

   Answer.

   ○ $f(n) = O(g(n))$
   [✓ $f(n) = \Omega(g(n))$]
   ○ $f(n) = \Theta(g(n))$
   ○ None of the above.

7. What is the running time of Insertion Sort (for sorting in ascending order) on a list with the first $n - \lceil\sqrt{n}\rceil$ elements sorted in ascending order?

   Answer.

   ○ $O(\sqrt{n})$
   ○ $O(n \log n)$
   ○ $O(n^2)$
   [✓ $O(n^{3/2})$]
   ○ $O(n)$

Name:	**PSU Access ID:**

8. If $T(n) = 2T(\frac{n}{5}) + O(n)$, which of the following is true?

   Answer.

   ◯ $T(n) = O(n^{\log_5 2})$
   ◯ $T(n) = O(n^{\log_2 5})$
   [✓ $T(n) = O(n)$]
   ◯ $T(n) = O(n \log n)$
   ◯ None of the above

9. If $T(n) = 12T(\frac{n}{7}) + O(n^2)$, which of the following is true?

   Answer.

   [✓ $T(n) = O(n^2)$]
   ◯ $T(n) = O(n^2 \log n)$
   ◯ $T(n) = O(n \log n)$
   ◯ $T(n) = O(n^{\log_7 12})$
   ◯ None of the above

Name:	**PSU Access ID (xyz1234):**

True/False *(20 points)*

True or false? Fill in the correct bubble. No justification is needed. No points will be subtracted for wrong answers, so it is in your best interest to guess all you want.

T **F**

○ ○ $n^{0.5} = O((\log n)^{10})$.
$\boxed{\text{False}}$

○ ○ $n^3 2^n = O(4^n)$.
$\boxed{\text{True}}$

○ ○ $n^{\log n} = O(2^n)$.
$\boxed{\text{True}}$

○ ○ If $f(n)$ and $g(n)$ are non-negative functions, then either $f(n) = O(g(n))$ or $g(n) = O(f(n))$.
$\boxed{\text{False}}$ **Explanation:** Consider $f(n) = 1$ when n is even and n^2 when n is odd. Let $g(n) = n$. Then, we can see that $f(n) \notin O(g(n))$ and $g(n) \notin O(f(n))$.

○ ○ If $f(n) = O(g(n))$, then there is some value of n where $f(n) \geq g(n)$.
$\boxed{\text{False}}$ **Explanation:** Consider $f(n) = n$ and $g(n) = n + 1$. Clearly $f(n) = O(g(n))$. But there is no value of n for which $f(n) \geq g(n)$.

○ ○ The procedure Heapify-Down could make $O(1)$ swaps only.
$\boxed{\text{True}}$ **Explanation:** This asks about a possible case and not a general case.

○ ○ In a min heap, the largest element will be stored in the last position of the array.
$\boxed{\text{False}}$

○ ○ In a min heap, the left child of the root is smaller than the right child of the root.
$\boxed{\text{False}}$ **Explanation:** The min-heap property does not specify a relationship between the left and right children. There is no order between the left and right children of each node.

○ ○ Checking whether a min heap with n elements contains a certain element takes $O(\log n)$ time.
$\boxed{\text{False}}$

○ ○ $\sum_{i=1}^{n} 4^i = O(4^n)$.
$\boxed{\text{True}}$

Name:	**PSU Access ID:**

Recurrences *(12 points)*

Each of the following scenarios outlines a divide-and-conquer algorithm. In each case, write down the appropriate recurrence relation for the running time as a function of the input size n and give its solution. Giving your solution in O-notation suffices. You need not give a full derivation of your solution, but you should indicate how you arrived at it (e.g., by using the Master Theorem). You may assume that n is of some special form (e.g., a power of two or of some other number), and that the recurrence has a convenient base case with cost $\Theta(1)$.

(a) An input of size n is broken down into 16 subproblems, each of size $n/4$. The time taken to construct the subproblems, and to combine their solutions, is $\Theta(n^2)$.

Solution: In this case, $T(n) = 16T(n/4) + O(n^2)$. Using the Master Theorem ($a = 16, b = 4, d = 2$), we get $\log_b a = \log_4 16 = 2 = d$, so $T(n) = O(n^2 \log n)$.

(b) An input of size n is broken down into $\sqrt{n}$ subproblems, each of size $\sqrt{n}$. The time taken to construct the subproblems, and to combine their solutions, is n.

Solution: $T(n) = \sqrt{n}T(\sqrt{n}) + n$. This recurrence was essentially Problem 5j from Homework 3, and if you remembered the solution, it was fine to justify as "this was a HW problem". Alternatively, you can note by unfolding that

$$T(n) = n^{\frac{1}{2}+\frac{1}{4}+\cdots+\frac{1}{2^k}} + kn = T(n^{\frac{1}{2^k}}) + kn$$

Now if we let $k = \log_2 \log_2 n$, we have $n^{\frac{1}{2^k}} = 2$ which is constant. Since $\frac{1}{2} \leq \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^k} \leq 1$, we have $T(n) = \Theta(n \log \log n)$.

Name:	**PSU Access ID:**

Algorithms. *(15 points)*

Given an array of n distinct integers and two numbers i and j such that $1 \leq i \leq j \leq n$, design an algorithm that finds the sum of all elements between the i-th and the j-th smallest elements of the array. For example, if the array contains the numbers $20, 8, 22, 4, 12, 10, 14$, $i = 3$ and $j = 6$, then the output should be 56 (the third smallest integer in the array is 10 and the sixth smallest is 20, so $10 + 12 + 14 + 20 = 56$).

Design and algorithm for this task with $O(n)$ running time. You may assume that $j = O(\sqrt{n})$. For full credit, make sure to include an explanation of how and why your algorithm works and a run time analysis.

Solution: The central idea of the solution is to use a min-heap to retrieve the smallest numbers from the input array. This idea was also used in Worksheet 3 Problem 5. As discussed in class, the Build-Heap operation to create a min-heap from the input array runs in $O(n)$. Deleting the minimum element from this min-heap is an $O(\log n)$ operation, and we are given $j = O(\sqrt{n})$. Therefore, deleting the j smallest elements takes $O(\sqrt{n} \log n)$. We can split these j deletions into a first group of $i - 1$ deletions (the elements that are too small to be included in the final sum) and a second group of $j - i + 1$) deletions (the elements we include in the final sum). We keep a variable for the running sum of every element in the second group, which will be our output. Elements greater than the j smallest simply remain in the heap and are not considered.

```
Data: An array of $n$ distinct integers $A$
Result: The sum of all elements between the $i$-th and $j$-th smallest
Build-Heap($A$);
for $k$ from 1 to i-1 do
    Delete($A$, 0);
end
$s = 0$;
for $k$ from $i$ to $j$ do
    $s = s$+GetMin($A$);
    Delete($A$, 0);
end
return s;
```

This is just one possible way to write the idea of using a heap in pseudocode. The running time for the above algorithm is $O(n) + O(\sqrt{n} \log n) = O(n)$.

An equivalent way to phrase the solution is to build the min-heap, then run the first j iterations of Heapsort. This will sort the j smallest elements in descending order and place them at the end of the array, after which we iterate between elements $n - j$ and $n - i$ to calculate the final sum.

Note: It is even possible to solve this problem in $O(n)$ if we remove the $j = O(\sqrt{n})$ constraint. Though the heap approach would be too slow in this case, there is an $O(n)$ algorithm for calculating the k-th smallest number in an unsorted array called the Median-of-Medians algorithm. Using this algorithm to find the i-th and j-th smallest numbers, the sum could then be calculated by iterating over the entire array and adding any value between them to the final sum.

Name:	**PSU Access ID (xyz1234):**

Name:	PSU Access ID (xyz1234):

Exam 2 - A

Name:

Penn State access ID (xyz1234) in the following box:

Student ID number (9XXXXXXXX):

TA and/or section time:

Instructions:

- Answer all questions. Read them carefully first. Be precise and concise. Handwriting needs to be neat. Box numerical final answers.
- Please clearly write your name and your PSU access ID (i.e., xyz1234) in the box on top of **every page**.
- Write in only the space provided. You may use the back of the pages only as scratch paper. **Do not write your solutions in the back of pages!**
- **Do not write outside of the black bounding box**: the scanner will not be able to read anything outside of this box.
- We are providing two extra page at the end if you need extra space. Make sure you mention in the space provided that your answer continues there.

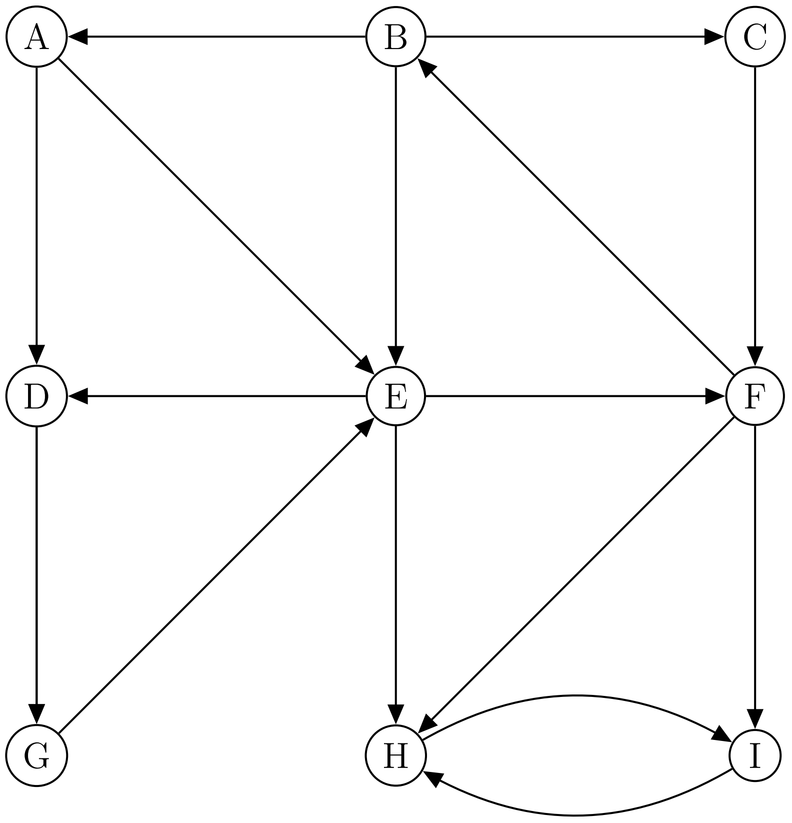
Good luck!

Name:	**PSU Access ID (xyz1234):**

Multiple choice questions *(26 points)*

For each of the following questions, select the right answer by filling the corresponding grading bubble grading bubbles.

1. Do a depth-first search of the graph below. Process vertices (and edges out of a vertex) in lexicographical order.



(a) The pre-visit number of node A is:
[✓ 1] ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

(b) The pre-visit number of node B is:
◯1 ◯2 ◯3 ◯4 ◯5 [✓ 6] ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

(c) The pre-visit number of node C is:
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 [✓ 7] ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

Name:	**PSU Access ID:**

(d) The pre-visit number of node D is:
◯1 [✓ 2] ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

(e) The pre-visit number of node E is:
◯1 ◯2 ◯3 [✓ 4] ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

(f) The pre-visit number of node F is:
◯1 ◯2 ◯3 ◯4 [✓ 5] ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

(g) The pre-visit number of node G is:
◯1 ◯2 [✓ 3] ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

(h) The pre-visit number of node H is:
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 [✓ 10] ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

(i) The pre-visit number of node I is:
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 [✓ 11] ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

(j) The post-visit number of node A is:
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 [✓ 18]

(k) The post-visit number of node B is:
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 [✓ 9] ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

(l) The post-visit number of node C is:
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 [✓ 8] ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

(m) The post-visit number of node D is:
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 [✓ 17] ◯18

(n) The post-visit number of node E is:
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 [✓ 15] ◯16 ◯17 ◯18

(o) The post-visit number of node F is:
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 [✓ 14] ◯15 ◯16 ◯17 ◯18

(p) The post-visit number of node G is:
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 [✓ 16] ◯17 ◯18

(q) The post-visit number of node H is:
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 [✓ 13] ◯14 ◯15 ◯16 ◯17 ◯18

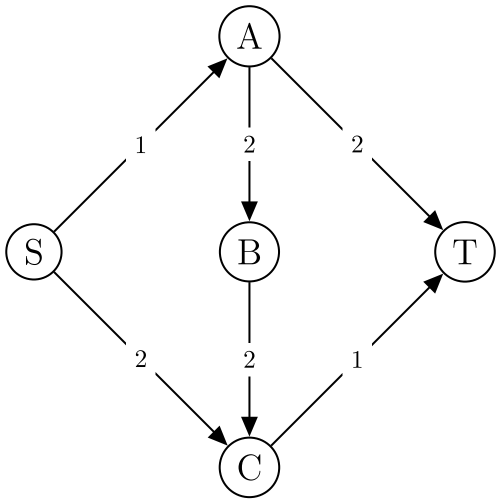
(r) The post-visit number of node I is:
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 [✓ 12] ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

Name:	**PSU Access ID (xyz1234):**

2. The strongly connected components of the graph in part (1.), in the order in which they would be discovered by the algorithm from class (DFS in the reversed graph, then DFS in the graph), will be:

   - ◯ HI, BCF, DEG, A
   - ◯ BCEF, HI, DEG, A
   - ◯ ABCDEFGH
   - [✓ HI, ABCDEFG]
   - ◯ A, BCF, DEG, HI
   - ◯ HI, ADGEFB, C
   - ◯ ABCDEFG, HI
   - ◯ EFHI, ADGEFB
   - ◯ A, DGE, BCF, HI

3. Consider the following flow network with source S and sink T.

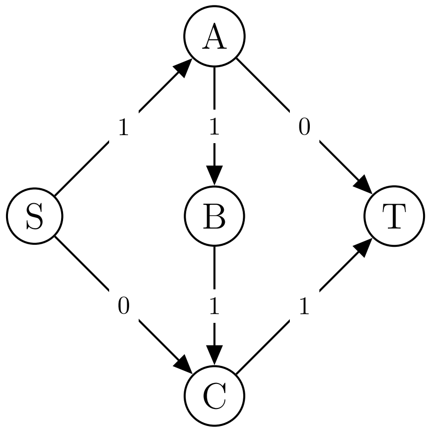


   The value of the maximum flow is:

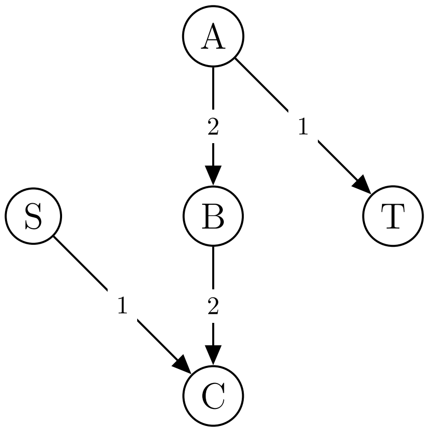
   - ◯ 4
   - ◯ 1
   - [✓ 2]
   - ◯ 3
   - ◯ 3/2

Name:	**PSU Access ID:**

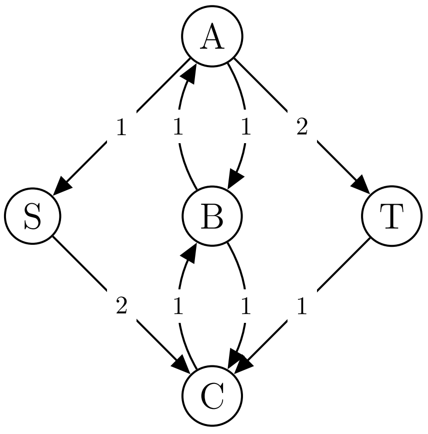
4. Suppose we run the Ford-Fulkerson algorithm on the flow network from the part 3 using depth-first search (and lexicographical order) to find augmenting S-T paths in the residual graphs in each iteration. Assuming the initial flow f_0 is the 0 flow, after two iterations of the Ford-Fulkerson algorithm, the residual graph G_{f_2} is:



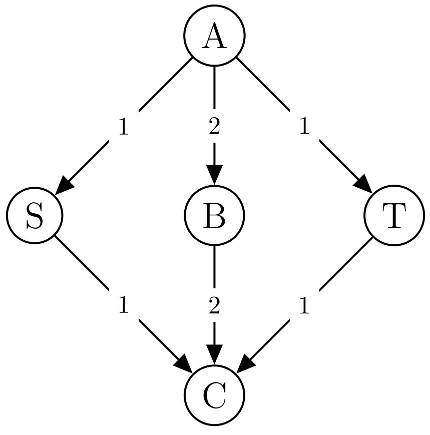
(a)



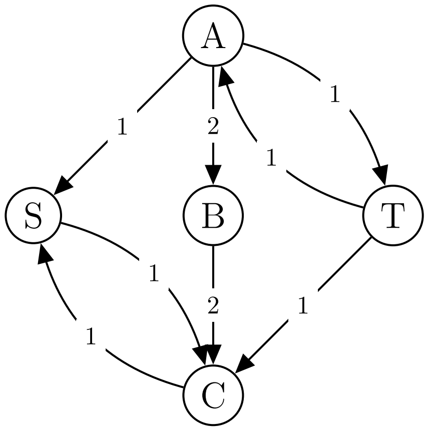
(b)



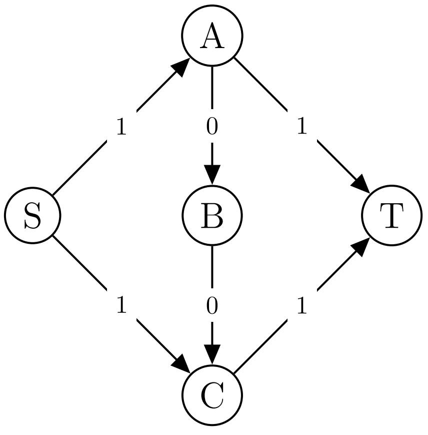
(c)



(d)



(e)



(f)

○ (a)
○ (b)
○ (c)
○ (d)
[✓ (e)]
○ (f)

Name:	**PSU Access ID:**

True/False *(20 points)*

True or false? Fill in the correct bubble. No justification is needed. Correct answers receive 2 pts; incorrect or blank answers receive 0 pts.

T **F**

1. ○ ● In a DFS on a directed graph $G = (V, E)$, if u was explored before v, and there is a path from u to v, then $post(v)$ is greater than $post(u)$.

Solution: Vertex u would still be in the call stack if there is a path from u to v. u will be popped only after v.

2. ○ ● There is no known polynomial-time algorithm to solve the maximum flow problem.

Solution: The Edmonds-Karp algorithm (mentioned in lecture) runs in polynomial time.

3. ● ● The longest path in a graph can be found by multiplying the cost of each edge by -1 and then running the Bellman-Ford algorithm.

Solution: We intended this question to be False, since multiplying by -1 could create negative cycles in the graph. For this, we would have needed to specify that the path was simple. We did not, so we gave credit to both T/F answers.

4. ● ○ In a weighted graph $G = (V, E)$ where all edge weights are positive integers and at most 1000, we can find the shortest paths from a given vertex s to any other vertex in $O(|V| + |E|)$ time.

Solution: Refer lecture notes: Split the weighted edges into dummy vertices with edge weights 1 and run BFS.

5. ● ○ Multiplying each edge weight in a graph by a constant $k > 0$ will not affect the shortest path of a graph.

Solution: The length of every path in the graph is scaled by the same factor.

6. ● ○ Given a flow network, let f be any flow and let (A, B) be an $s - t$ cut. Then, the value of f is always less than or equal to the capacity of (A, B).

Solution: The value of the flow is always less than or equal to the capacity of any s-t cut.

7. ○ ● Every directed acyclic graph (DAG) has exactly one topological ordering.

Solution: Consider the graph with two nodes and no edges; or with three nodes, A, B and, C with A and B connected to C.

8. ○ ● In DFS in a directed graph, the vertex with the largest post-visit number is in a sink strongly connected component.

Solution: Recall that there is not easy/efficient way of finding a vertex in a sink SCC; hence the need for the reverse graph algorithm, etc. The vertex with the largest post-visit number is in a source SCC.

9. ○ ● Suppose we increase the capacity of an edge in a minimum cut of a flow network. As a result, the value of the maximum flow also increases.

Solution: This is true only if we have a unique min cut. Counterexample: $S \xrightarrow{1} A \xrightarrow{1} T$

10. ◯ ● The Floyd-Warshall algorithm for all-pairs shortest paths works even when there are negative cycles.

Solution: The shortest path problem is not defined when there are negative cycles.

Name:	**PSU Access ID (xyz1234):**

Special vertex *(15 points)*

You are given a strongly connected directed graph $G = (V, E)$ with positive edge weights and a vertex $w \in V$. Give an algorithm for finding the shortest paths between all pairs of vertices of G (excluding w), with the restriction that these paths must all pass through w. Your algorithm should run in $O(|V|^2)$. Provide a justification of why your algorithm works and a run time analysis.

Solution:

This problem was included in Worksheet 7 (see Problem 5).

Let P be the shortest path from vertex u to v passing through w. Note that, between w and v, P must necessarily follow the shortest path from w to v. By the same reasoning, between u and w, P must follow the shortest path from w and u in the reverse graph. Both these paths are guaranteed to exist as the graph is strongly connected. Hence, the shortest path from u to v through w can be computed for all pairs u,v by performing two runs on Dijkstra's algorithm from w, one on the input graph G and the other on the reverse of G. The running time of reversing the graph is $O(|V| + |E|)$, and the running time of Dijkstra's algorithm is $O(|V| \log |V| + |E|)$. Both are dominated by looking up all the $O(|V^2|)$ pairs of distances.

Note that modifying Floyd-Warshall by removing the outer for-loop (the loop that determines which node the current shortest path must go through) leads to an incorrect shortest path calculation. To correctly find the shortest paths through node w, you must consider all the shortest paths through other nodes as well. For example, imagine that (u, w) is not an edge in the graph so that the shortest path from u to w requires going through some other series of nodes. Then running just the inner two for-loops of Floyd-Warshall going through w will not find the shortest path from u to w, and the implementation would output ∞ for the pair u, v.

Name:	**PSU Access ID (xyz1234):**

Rankings *(15 points)*

Let $G = (V, E)$ be a directed graph and suppose that each vertex of G has a positive integer weight. (You may assume that all weights are distinct and that the vertices are given to you sorted in increasing order by weight). For every vertex $v \in V$, you would like to find the vertex of minimum weight that is reachable from v. Provide an algorithm for this task and justify why it works. The running time of your algorithm should be $O(|V| + |E|)$.

Solution: This question uses the same ideas as in Homework 5, Question 5. (It is in fact the same question, except that the weights are not numbers from 1 to n, but they are given to us sorted.) The provided graph will be used in the same way as the graph constructed in part (a) of the homework, solving the question with the algorithm from part (b). If we reverse the direction of every edge in the graph, the set of vertices that can reach vertex v in G becomes precisely the set of vertices that vertex v can reach in G^R. Furthermore, all vertices within the same SCC should ultimately reach the same vertex of minimum weight. Thus, for all unvisited vertices in the reverse graph, we can run Explore starting from the lowest-weighted unvisited vertex i, and for all the vertices j that are reachable from i, we assign i to be vertex j's minimum weight vertex.

We can see that this algorithm is very similar to our algorithm for finding SCCs. Rather than starting from a vertex in a sink SCC, however, we start from the vertex with the smallest value. So rather than finding SCCs one at a time, we'll first find all the vertices in the SCCs that can reach smallest weight vertex (which we know, since we are provided sorted vertices), then all the vertices in SCCs that can reach the next smallest available weighted vertex, and so on.

Algorithm

Let G^R be the graph G with its edge directions reversed. The algorithm is as follows.

```
Function Rankings($G$):
    while there are unvisited vertices in $G^R$ do
        Run Explore on $G^R$ starting from the lowest-weighted unvisited node $i$
        for $j$ visited by this Explore do
            minimum_weight_vertex[$j$] := $i$
        end
    end
```

To see that this algorithm is correct, note that if a vertex i is assigned a value then that value is the smallest of the vertices that can reach it in G^R, and every vertex is assigned a value because the loop does not terminate until this happens. Now observe that the set of vertices reachable by i in G^R is the set of vertices which can reach i in G.

The running time is $O(|V| + |E|)$ since computing G^R can be done in linear time, and we process every vertex and edge $O(1)$ times in the DFS.

Note: There is an alternative solution that works for arbitrary, unsorted weights. The idea is to first use the SCC algorithm to construct a metagraph (it is a DAG), with each metanode having the minimum weighted vertex in the corresponding SCC (note that the SCC algorithm constructs a topologically sorted metagraph). Then, look at the children of each unvisited metanode of the metagraph, going in order from the metanode at the end of the topological sort to the beginning. When looking at the child metanodes, assign the minimum of the weight (and vertex) of the children and the current metanode as the new weight of the metanode (a tuple of weight value and node can be passed throughout the procedure). All vertices in a metanode are assigned the same minimum weight vertex.

Name:	**PSU Access ID (xyz1234):**

Name:	**PSU Access ID (xyz1234):**

Midterm 3 - Exam A

Name:

Penn State access ID (xyz1234) in the following box:

Student ID number (9XXXXXXXX):

Lecture section time:

Instructions:

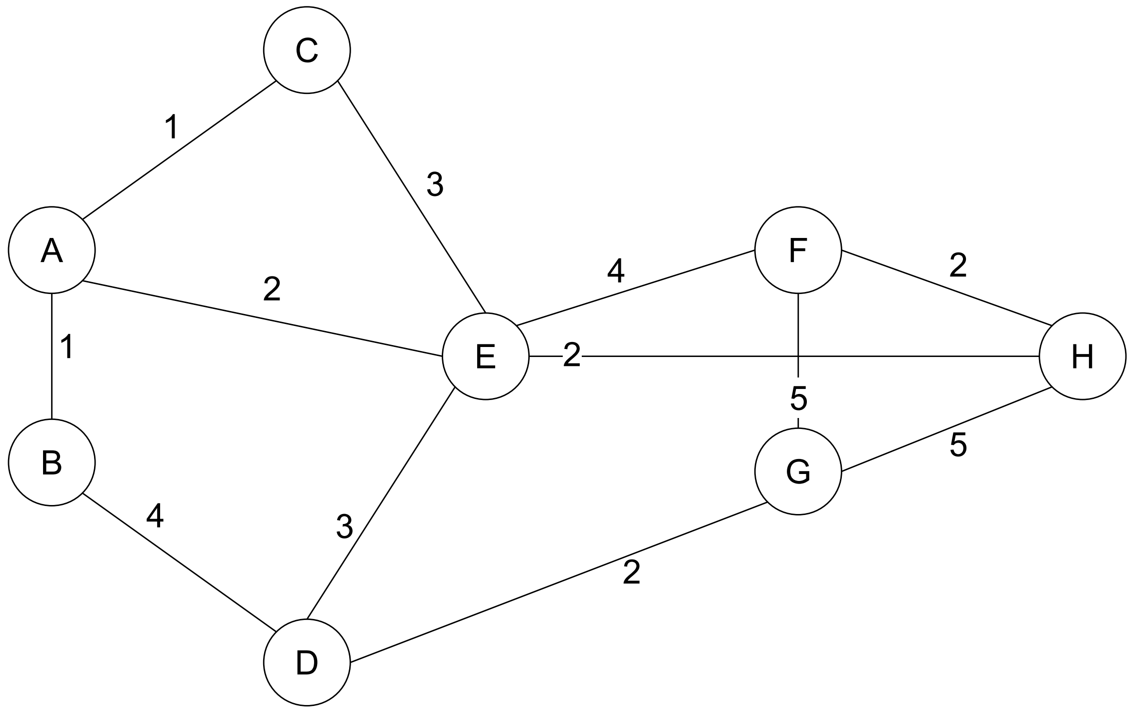
- Answer all questions. Read them carefully first. Be precise and concise. Handwriting needs to be neat. Box numerical final answers.
- Please clearly write your name and your PSU access ID (i.e., xyz1234) in the box on top of **every page**.
- Write in only the space provided. You may use the back of the pages only as scratch paper. **Do not write your solutions in the back of pages!**
- **Do not write outside of the black bounding box**: the scanner will not be able to read anything outside of this box.
- We are providing one extra page at the end if you need extra space. Make sure you mention in the space provided that you answer continues there.

Good luck!

Name:	**PSU Access ID (xyz1234):**

1 Kruskal's algorithm *(7 points)*

Use Kruskal's algorithm on the graph shown below and compute the minimum spanning tree. Write down an order in which edges were added to the tree.



Solution:
The Kruskal algorithm starts by sorting the edges in increasing order of their weights. It then iteratively adds the lowest weighted edge to the minimum spanning tree as long as it doesn't create a cycle, continuing this process until all nodes are included. The edges are added in the following order: AB, AC, AE, EH, FH, DG, ED.

Please note that the order of edges provided is one of the several valid solutions.

Name:	PSU Access ID (xyz1234):

Name:	**PSU Access ID (xyz1234):**

2 Prefix-free encodings *(7 points)*

Which of the following encodings are prefix-free? Note that it is possible that there are zero, one, or more than one prefix-free encodings.

a)
a $\rightarrow$ 1
b $\rightarrow$ 11
c $\rightarrow$ 101

Not Prefix-free

b)
a $\rightarrow$ 10
b $\rightarrow$ 11
c $\rightarrow$ 101

Not Prefix-free

c)
a $\rightarrow$ 1
b $\rightarrow$ 01
c $\rightarrow$ 001

Prefix Free

d)
a $\rightarrow$ 10
b $\rightarrow$ 11
c $\rightarrow$ 110
Not Prefix Free

Name:	**PSU Access ID (xyz1234):**

3 Huffman's algorithm *(7 points)*

Compute and draw a prefix-free encoding tree using Huffman algorithm for symbols a, b, c, d, e with the following multiplicities and write down encoding corresponding to each symbol:

$$a : 8, b : 1, c : 4, d : 3, e : 8$$

One possible solution will be:

$$e : 0, a : 10, c : 110, b : 1110, d : 1111$$

The total number of bits is $8 \times 1 + 8 \times 2 + 4 \times 3 + 1 \times 4 + 3 \times 4 = 52$

Name:	**PSU Access ID (xyz1234):**

4 Edit distance *(7 points)*

Compute the edit distance between strings S_1 and S_2 by filling out the table below using the dynamic programming procedure. Edit distance is the number of insertions, deletions, and substitutions that are needed to turn one string into another.

$$S_1 = AGCG, S_2 = TGC$$

	–	A	G	C	G
–					
T					
G					
C					

	-	**A**	**G**	**C**	**G**
-	0	1	2	3	4
T	1	1	2	3	4
G	2	2	1	2	3
C	3	3	2	1	2

Name: **PSU Access ID (xyz1234):**

5 True/False *(20 points)*

True or false? Fill in the right bubble. No justification is needed. No points will be subtracted for wrong answers, so it is to your best interest to guess all you want.

T ○ **F** ○ 1. The minimum spanning tree can be computed using a greedy algorithm.
True

T ○ **F** ○ 2. Each graph has only one minimum spanning tree.
False

T ○ **F** ○ 3. A minimum spanning tree of a connected graph is connected.
True

T ○ **F** ○ 4. Disjoint sets data structure describes a procedure decomposing a set into two subsets.
False

T ○ **F** ○ 5. Huffman algorithm computes fixed-length encoding for each symbol.
False

T ○ **F** ○ 6. Huffman algorithm assigns the shortest encoding to the rarest symbol.
False

T ○ **F** ○ 7. The set cover problem maximizes the number of selected sets.
False

T ○ **F** ○ 8. Both greedy algorithms and dynamic programming decompose a problem into subproblems.
True

T ○ **F** ○ 9. Edit distance of two strings of lengths M and N can be computed in $O(min\{M, N\})$ running time.
False

T ○ **F** ○ 10. Floyd-Warshall algorithm can be reformulated as a dynamic programming procedure.
True

Name:	**PSU Access ID (xyz1234):**

6 Choosing non-overlapping segments *(15 points)*

Given n segments $[s_i, f_i]$, where s_i, f_i are positive real numbers and $s_i < f_i$ for $i = 1, \ldots, n$. Segments $[s_i, f_i]$ and $[s_j, f_j]$ overlap if there exists a number X that belongs to both segments:

- $s_i \leq X \leq f_i$,
- $s_j \leq X \leq f_j$.

We call a subset S of segments non-overlapping if no two segments in S overlap. Our goal is to find a non-overlapping subset with the maximum number of segments. Does the greedy strategy described below provide an optimal solution to the problem? If not, provide a counterexample. If so, justify.

The greedy strategy: Choose the segment x that ends last, discard all segments that overlap with x, and repeat the procedure with remaining segments.

Solution: The greedy strategy does not provide an optimal solution. Consider segments $[1, 10]$, $[2, 3]$, and $[4, 5]$. The segment ending last is $[1, 10]$, which overlaps with both $[2, 3]$, and $[4, 5]$, so they are both discarded, leaving $[1, 10]$, but the optimal solution is $[2, 3]$ and $[4, 5]$.

Name:	**PSU Access ID (xyz1234):**

Name:	**PSU Access ID (xyz1234):**

7 Computing a dense subsequence *(15 points)*

Let A be a sequence of integers $a_1, \ldots, a_n$. A subsequence A' is *dense* if, for each pair of consecutive elements (a_i, a_{i+1}) in A, at least one of them is in A'. Describe a dynamic programming algorithm that computes a dense subsequence with the largest sum of its elements.

Solution:
Intuition: we must choose whether to include or exclude every element a_i from the sum. However, the dynamic program does not need to explicitly pick one of those two options, it can compute the maximum sum up to element a_i for both cases and use that information in future iterations. When computing the maximum sums for the next index, we can force our subsequence to be dense with the previous sum that was forced to use its element. If we choose to exclude a_{i+1}, we must include a_i, but if we include a_{i+1}, we can include or exclude a_i. Note that we can ignore a_{i+2} while computing the sum up to a_{i+1} since we compute both the "forced to take a_{i+1}" value and the "free to choose" value, which can be used as needed on the a_{i+2} iteration, and so on.

Subproblems: Let $f[i]$ be the largest sum achievable using elements from a_1 to a_i, inclusive, where we are forced to use element a_i when computing $f[i]$. Let $c[i]$ be the largest sum achievable using elements from a_1 to a_i, inclusive, where we are free to choose whether to include a_i or not. Note that these sums could also be stored in tuples in a single array, having two arrays is not a fundamental change to the dynamic programming paradigm.

Base Case: $f[1] = a_1$, since it is forced to use a_1. $c[1] = \max(a_1, 0)$, since if a_1 is negative, we are free to exclude it.

Recurrence Relation: $f[i] = a_i + c[i-1]$ and $c[i] = \max(a_i + c[i-1], f[i-1])$. For computing $f[i]$, note that $c[i-1] \geq f[i-1]$, since $c[i-1]$ is recursively computed as the maximum of $a_{i-1} + c[i-2] = f[i-1]$ and $f[i-2]$ (or $f[1]$ and 0 in the base case). Therefore, as $f[i]$ is forced to use a_i, the maximum sum it can achieve is given by the above formula. As for $c[i]$, there are two options, of which we take the maximum. We either include a_i and take the maximum sum up to a_{i-1} with the option to exclude a_{i-1}, or we exclude a_i and force ourselves to include a_{i-1} by using $f[i-1]$.

The final sum will be the value of $c[n]$ after the iterations are complete. This is the maximum sum achievable through any dense combination of inclusions and exclusions, where we have not forced the final element to be included.

Running Time: This solution uses two length-n arrays, with constant-time operations to compute each cell of each array. Therefore, the running time is $O(n)$.

Alternate Solution #1:
Another solution is to use a single array f where $f[i]$ is the maximum sum using elements a_1 to a_i and forced to include a_i. The base cases for this solution are $f[1] = a_1$ and $f[2] = \max(a_2, a_1 + a_2)$. Then the recurrence relation is $f[i] = \max(a_i + f[i-1], a_i + f[i-2])$ and the final answer is $\max(f[n], f[n-1])$.

Alternate Solution #2:
Another solution is to replace array c with an array e such that $e[i]$ is the maximum sum using elements a_1 to a_i and forced to exclude a_i. The base cases for this solution are $f[1] = a_1$ and $e[1] = 0$. Then the recurrence relations are $f[i] = \max(a_i + f[i-1], a_i + e[i-1])$ and $e[i] = f[i-1]$. The final answer is $\max(f[n], e[n])$.

Name:	PSU Access ID:

Name:	**PSU Access ID (xyz1234):**

Final Exam

Name:

Penn State access ID (xyz1234) in the following box:

Student ID number (9XXXXXXXX):

TA and/or section time:

Instructions:

- Answer all questions. Read them carefully first. Be precise and concise. Handwriting needs to be neat. Box numerical final answers.
- Please clearly write your name and your PSU access ID (i.e., xyz1234) in the box on top of **every page**.
- Write in only the space provided. You may use the back of the pages only as scratch paper.
- **Do not write outside of the black bounding box**: the scanner will not be able to read anything outside of this box.
- We are providing four extra pages at the end if you need extra space. Make sure you mention in the space provided that your answer continues there.

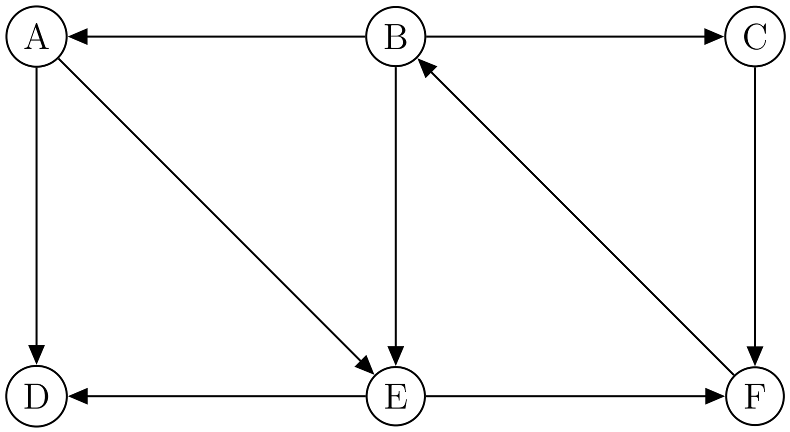
Good luck!

Name:	**PSU Access ID (xyz1234):**

Multiple choice questions *(40 points)*

Select the correct answer for each of the following questions by filling in the corresponding grading bubble grading bubbles.

1. (7 pts) Do a depth-first search of the graph below. Process vertices (and edges out of a vertex) in lexicographical order.



(a) The pre-visit number of node B is:
○1 ○2 ○3 ○4 ○5 [✓ 6] ○7 ○8 ○9 ○10 ○11 ○12

(b) The pre-visit number of node D is:
○1 [✓ 2] ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 ○11 ○12

(c) The pre-visit number of node C is:
○1 ○2 ○3 ○4 ○5 ○6 [✓ 7] ○8 ○9 ○10 ○11 ○12

(d) The pre-visit number of node F is:
○1 ○2 ○3 ○4 [✓ 5] ○6 ○7 ○8 ○9 ○10 ○11 ○12

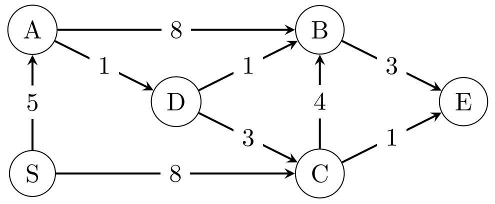
(e) The post-visit number of node D is:
○1 ○2 [✓ 3] ○4 ○5 ○6 ○7 ○8 ○9 ○10 ○11 ○12

(f) The post-visit number of node B is:
○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 [✓ 9] ○10 ○11 ○12

(g) The post-visit number of node C is:
○1 ○2 ○3 ○4 ○5 ○6 ○7 [✓ 8] ○9 ○10 ○11 ○12

Name:	**PSU Access ID:**

2. (7 pts) Run the priority queue implementation of Dijkstra's algorithm in the graph below from node S. Consider the shortest path distances from S to each other vertex obtained after each iteration of the algorithm (you may keep track of them using the table below).



Auxiliary table:

Iteration	S	A	B	C	D	E
0	0	∞	∞	∞	∞	∞
1				x_1		
2			x_2		x_3	
3			x_4			x_5
4						
5					x_6	x_7
6						

(a) The value of x_1 is:
◯3 ◯4 ◯5 ◯6 ◯7 [✓ 8] ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 ◯19 ◯20 ◯∞

(b) The value of x_2 is:
◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 [✓ 13] ◯14 ◯15 ◯16 ◯17 ◯18 ◯19 ◯20 ◯∞

(c) The value of x_3 is:
◯3 ◯4 ◯5 [✓ 6] ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 ◯19 ◯20 ◯∞

(d) The value of x_4 is:
◯3 ◯4 ◯5 ◯6 [✓ 7] ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 ◯19 ◯20 ◯∞

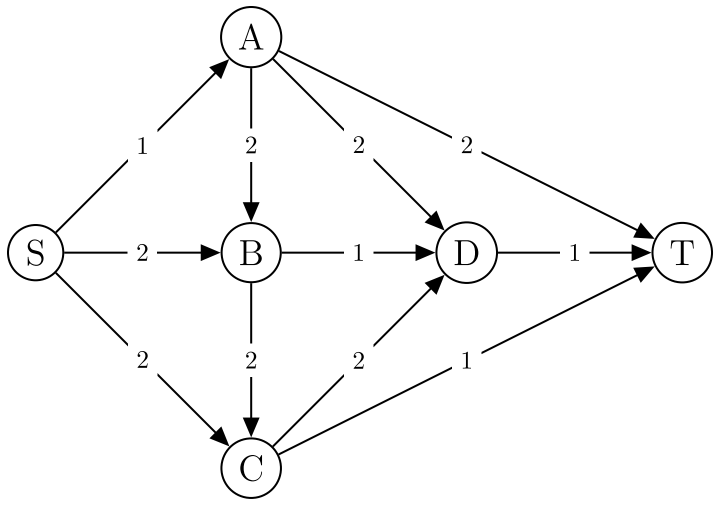
(e) The value of x_5 is:
◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 ◯19 ◯20 [✓ ∞]

(f) The value of x_6 is:
◯3 ◯4 ◯5 [✓ 6] ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 ◯19 ◯20 ◯∞

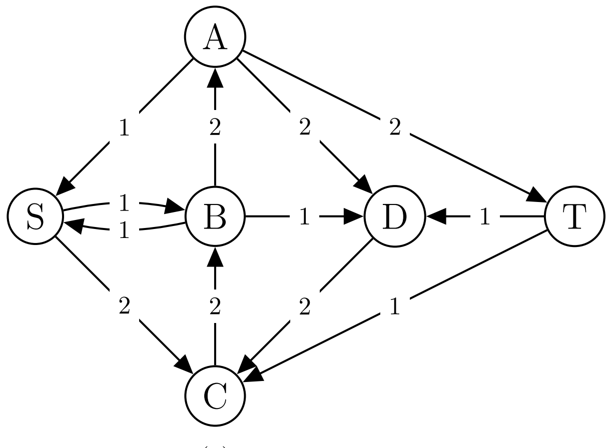
(g) The value of x_7 is:
◯3 ◯4 ◯5 ◯6 ◯7 ◯8 [✓ 9] ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 ◯19 ◯20 ◯∞

Name:	**PSU Access ID (xyz1234):**

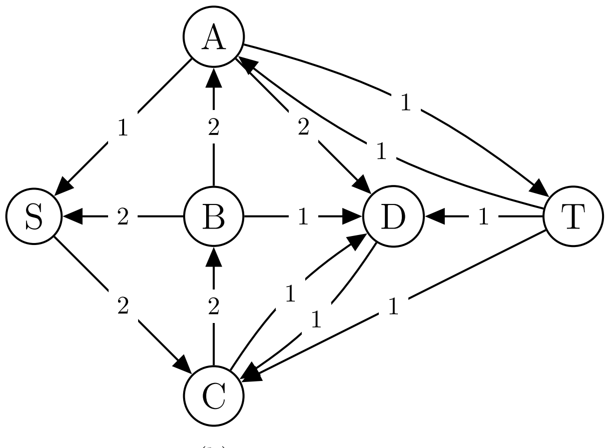
3. (7 pts) Consider the following flow network with source S and sink T.



Suppose we run the Ford-Fulkerson algorithm on this flow network using depth-first search (and lexicographical order) to find augmenting S-T paths in the residual graphs in each iteration. Assuming the initial flow f_0 is the 0 flow, after two iterations of the Ford-Fulkerson algorithm, the residual graph G_{f_2} is:



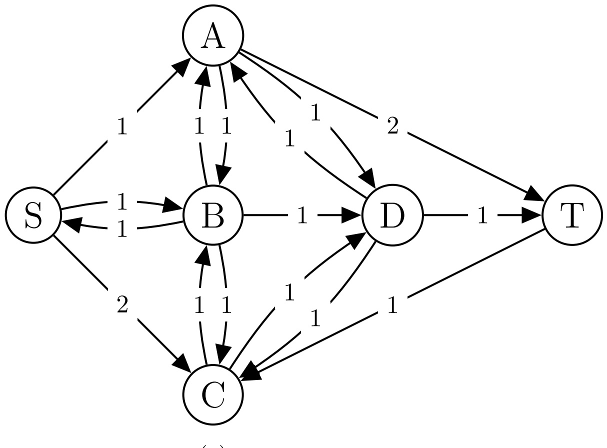
(a)



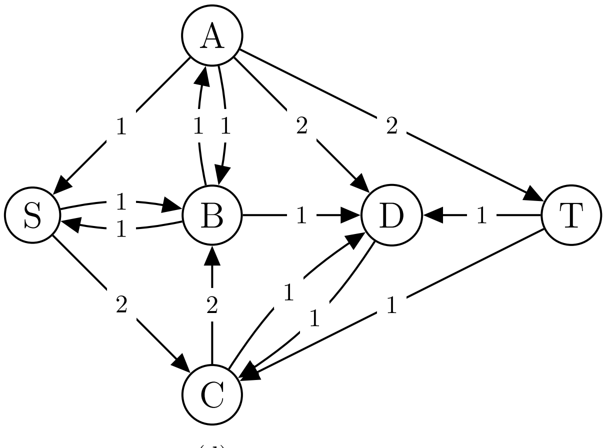
(b)

Note: More options and answer bubbles on the next page!!

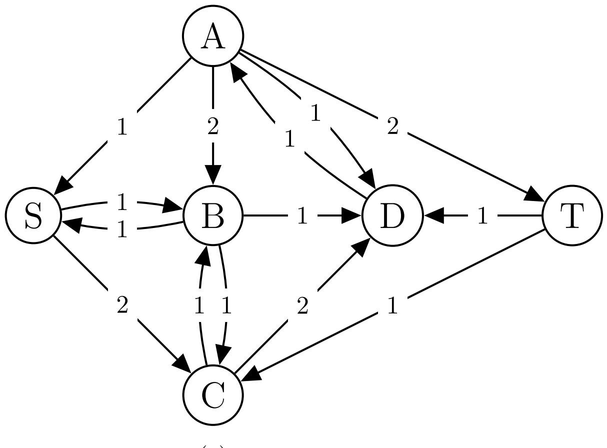
Name:	**PSU Access ID (xyz1234):**



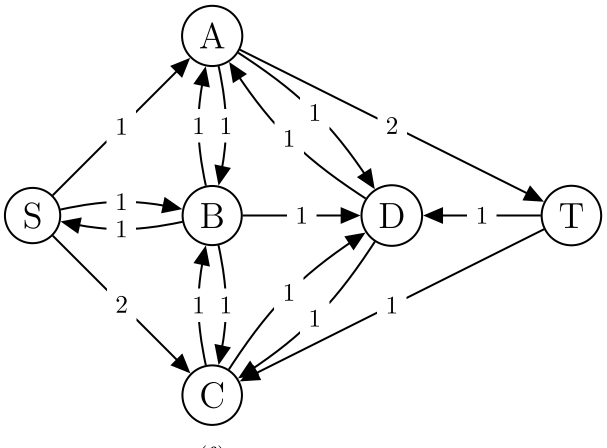
(c)



(d)



(e)



(f)

- ◯ (a)
- ◯ (b)
- ◯ (c)
- ◯ (d)
- ● (e)
- ◯ (f)
- ◯ none of the above

Name:	**PSU Access ID (xyz1234):**

4. (2 pts) If $T(n) = 25T(\frac{n}{5}) + O(n^2)$, which of the following is true?

   Answer.

   ◯ $T(n) = O(n^{\log_{25} 5})$
   ◯ $T(n) = O(n^2)$
   [✓ $T(n) = O(n^2 \log n)$]
   ◯ $T(n) = O(n \log n)$
   ◯ None of the above

5. (2 pt) An unordered array contains n distinct elements. The number of comparisons needed to find any number in the array that is not its minimum or its maximum is:

   Answer.

   ◯ $\Theta(n \log n)$
   ◯ $\Theta(n)$
   [✓ $\Theta(1)$]
   ◯ $\Theta(\log n)$
   ◯ None of the above

Name: **PSU Access ID (xyz1234):**

6. (9 pts) We define the fitting edit distance FE between strings s_1 and s_2 of lengths M and N, respectively, as the minimum edit distance between s_1 and a substring of s_2. For example, $FE(AGG, TCACG) = 1$:

$$\begin{array}{ccccc} - & - & \text{A} & \text{G} & \text{G} \\ - & - & | & . & | \\ \text{T} & \text{C} & \text{A} & \text{C} & \text{G} \end{array}$$

To compute FE, we use a table T with $M+1$ columns and $N+1$ rows. $T(i,j)$ refers to a cell located at the intersection of i-th row and j-th column.

Below, choose correct components of a dynamic programming procedure computing FE using the table T:

(a) To fill out the first column and the first row, we define base cases as:
Answer.

- ○ $T(0,i) = 1,\ i = 0,\dots,M$ and $T(j,0) = 1,\ j = 0,\dots,N$
- ○ $T(0,i) = 0,\ i = 0,\dots,M$ and $T(j,0) = 0,\ j = 0,\dots,N$
- ○ $T(0,i) = 0,\ i = 0,\dots,M$ and $T(j,0) = j,\ j = 0,\dots,N$
- [✓ $T(0,i) = i,\ i = 0,\dots,M$ and $T(j,0) = 0,\ j = 0,\dots,N$]
- ○ $T(0,i) = i,\ i = 0,\dots,M$ and $T(j,0) = j,\ j = 0,\dots,N$
- ○ $T(i,i) = i,\ i = 0,\dots,min(M,N)$ and $T(j,j) = j,\ j = 0,\dots,min(M,N)$
- ○ $T(0,i) = j,\ i = 0,\dots,M$ and $T(j,0) = i,\ j = 0,\dots,N$

(b) To compute the remaining elements in the table, we define the recurrence formula as:
Answer.

- ○ $T(i,j) = min\{1 + T(i-1,j), diff(i,j) + T(i-1,j-1)\}$
- [✓ $T(i,j) = min\{1 + T(i-1,j), 1 + T(i,j-1), diff(i,j) + T(i-1,j-1)\}$]
- ○ $T(i,j) = min\{T(i-1,j), 1 + T(i,j-1), diff(i,j) + T(i-1,j-1)\}$
- ○ $T(i,j) = min\{1 + T(i-1,j), T(i,j-1), diff(i,j) + T(i-1,j-1)\}$
- ○ $T(i,j) = min\{1 + T(i-1,j), 1 + T(i,j-1), diff(i,j) + T(i-1,j-1), 0\}$
- ○ $T(i,j) = min\{1 + T(i,j-1), diff(i,j) + T(i-1,j-1), 0\}$
- ○ $T(i,j) = min\{T(i-1,j), diff(i,j) + T(i-1,j-1)\}$

where $diff(i,j) = 0$ if $x_i = y_j$ and 1 otherwise.

Name:	**PSU Access ID (xyz1234):**

(c) To compute the value of the fitting edit distance, find:
Answer.

- ◯ the value in $T(N, M)$,
- [✓ the minimum value in the last column,]
- ◯ the minimum value in the first row,
- ◯ the average of the last row,
- ◯ the minimum value in the last row,
- ◯ the minimum value in the whole table.

Name:	**PSU Access ID (xyz1234):**

7. (2 pts) If the primal linear programming (LP) problem is unbounded, then the corresponding dual LP problem is always:
Answer.

   - ◯ feasible,
   - [✓ infeasible,]
   - ◯ unbounded,
   - ◯ none of the above.

8. (2 pts) The decision version of the vertex cover problem asks if a graph has a vertex cover of size k. What is a certificate for this problem?
Answer.

   - ◯ a set of vertices of size k,
   - ◯ a vertex cover of the minimum size,
   - [✓ a vertex cover of size k,]
   - ◯ none of the above.

9. (2 pts) A problem X is NP-complete if and only if:
Answer.

   - ◯ no polynomial algorithm can solve X,
   - ◯ X has an efficient certifier,
   - ◯ X can be reduced to any NP problem in polynomial time,
   - ◯ any NP problem can be reduced to X in polynomial time,
   - [✓ none of the above.]

Name:	**PSU Access ID:**

True/False *(26 points)*

True or false? Fill in the correct bubble. No justification is needed. Correct answers receive 2 pts; incorrect or blank answers receive 0 pts.

	T	**F**	
1.	○	●	$\sqrt{n} = O((\log n)^{10})$
2.	●	●	The best known algorithm to multiply two $n \times n$ matrices takes $O(n^3)$ time.
3.	○	●	The best known algorithm to multiply two n-bit integers takes $O(n^{\log_2 3})$ time.
4.	○	●	The running time of the Bellman-Ford algorithm for all-pairs shortest paths in a graph $G = (V, E)$ is $O(\|V\|\|E\|)$.
5.	●	○	$\log(n!) = \Theta(n \log n)$
6.	○	●	Checking whether a min heap with n elements contains a certain element takes $O(\log n)$ time.
7.	○	●	If $f(n) = O(g(n))$, then $2^{f(n)} = O(2^{g(n)})$.
8.	○	●	Computing the longest common subsequence of two strings with lengths M and N takes $O(max(M, N))$ time.
9.	○	●	In a prefix-free encoding tree computed using the Huffman algorithm, the shortest encoding is assigned to the rarest symbol.
10.	●	○	The problem of finding the minimum spanning tree is in NP.
11.	●	○	The primal and dual LP problems always have the same optimal values of the objective functions.
12.	○	●	The simplex method always makes as few iterations as possible.
13.	○	●	If a problem in NP can be solved in polynomial time, then $P = NP$.

Name:	**PSU Access ID (xyz1234):**

Linear Programming *(30 points)*

Given a linear programming problem with constraints s_1, s_2, s_3 and decision variables x, y. Convert the problem to the 3rd standard form by introducing slack and/or surplus variables a_1, a_2, a_3 and solve it using the simplex method by filling out the tables below. While finding the pivot column, break ties by choosing the leftmost column. While finding the pivot row, break ties by choosing the topmost row.

$max\ x + 2y$

$s.t.$
$x \leq 6\ (s_1)$
$y - x \leq 2\ (s_2)$
$x + y \geq 4\ (s_3)$
$x, y \geq 0$

Name: **PSU Access ID (xyz1234):**

	x	y	a_1	a_2	a_3	b	
s_1	1	0	1	0	0	6	
s_2	-1	1	0	1	0	2	
s_3	1	1	0	0	-1	4	
z	-1	-2	0	0	0	0	

Initial tableau

	x	y	a_1	a_2	a_3	b	
s_1	0	-1	1	0	1	2	
s_2	0	2	0	1	-1	6	
s_3	1	1	0	0	-1	4	
z	0	-1	0	0	-1	4	

Iteration 1

	x	y	a_1	a_2	a_3	b	
s_1	0	0	1	1/2	1/2	5	
s_2	0	1	0	1/2	-1/2	3	
s_3	1	0	0	-1/2	-1/2	1	
z	0	0	0	1/2	-3/2	7	

Iteration 2

	x	y	a_1	a_2	a_3	b	
s_1	0	0	2	1	1	10	
s_2	0	1	1	1	0	8	
s_3	1	0	1	0	0	6	
z	0	0	3	2	0	22	

Iteration 3

Name:	**PSU Access ID (xyz1234):**

Answer multi-choice questions below.

1. (5 pts) Is the two-phase simplex method needed to solve this problem?
   Answer.

   [✓ Yes]
   No

2. (5 pts) What is the solution to the linear programming problem?
   Answer.

   6
   7
   [✓ 22]
   30
   None of the above

3. (5 pts) What are basic variables after the iteration 2?
   Answer.

   a_1, a_2, a_3
   x, a_1, a_2
   [✓ x, y, a_1]
   x, y, a_2
   None of the above

   More questions on the next page!

Name:	**PSU Access ID (xyz1234):**

4. (5 pts) What are the values of the decision variables after the final iteration?
Answer.

- ○ 4, 1
- ○ 1, 3
- [✓ 6, 8]
- ○ 12, 9
- ○ None of the above

5. (5 pts) What is the smallest positive coefficient in z row of the final tableau?
Answer.

- ○ 1
- [✓ 2]
- ○ 3
- ○ 4
- ○ None of the above

6. (5 pts) Construct the dual LP problem using new decision variables y_1, y_2, y_3 corresponding to the primal constraints s_1, s_2, s_3, respectively. Which of the following constraints will be in the dual LP problem? Mark all correct options.
Answer.

- [✓ $y_1 - y_2 + y_3 \geq 1$]
- ○ $y_1 + 2y_3 \geq 0$
- [✓ $y_2 + y_3 \geq 2$]
- ○ $6y_1 + 2y_2 + 4y_3 \leq 0$

Name:	**PSU Access ID (xyz1234):**

Second Best MST *(15 points)*

For a graph $G = (V, E)$, the second-best Minimum Spanning Tree is a spanning tree with the second minimum weight sum of all edges out of all spanning trees of graph G. Design an algorithm to find the second-best Minimum Spanning Tree of a given graph G. Include a justification of why your algorithm is correct and a run-time analysis.

Note: For full-credit your algorithm should run in $O(|V|^3)$ time or better.

Solution: First, use Kruskal's algorithm to find the Minimum Spanning Tree (MST), which takes $O(|E| \log |V|)$ time. For each edge not in the MST, add it to the MST, forming a cycle. Remove the heaviest edge from this cycle using Depth-First Search (DFS), which is efficient in $O(|V|)$ time for a tree+1 edge. Recalculate the cost for each new MST formed. There are $O(|E|)$ such MSTs and select the one with minimum cost in $O(|E|)$ time. The total runtime is $O(|E| \log |V|) + O(|E| \cdot |V|)$. In the worst-case scenario, where $|E| = O(|V|^2)$, the runtime becomes $O(|V|^3)$.

Name:	PSU Access ID (xyz1234):

Name:	**PSU Access ID (xyz1234):**

Name:	PSU Access ID (xyz1234):

Name:	**PSU Access ID (xyz1234):**